# Hands-on – The Hello World! Program

Cell Programming Workshop
Cell/Quasar Ecosystem Solutions Enablement

# Class Objectives

- **You will learn how to write, build and run "Hello World!" on the Cell System Simulator**
- **Navigate through the basic build process and make files**
- **Familiarize with gcc and xlc compilers**
- **Familiarize with the system simulator**
- **There are three different versions of "Hello World!" used in this session**
  - PPE only,
  - SPE only, and
  - Cell BE, i.e. using both PPE and SPE
    - Synchronous
    - Asynchronous

**Trademarks** - Cell Broadband Engine ™ is a trademark of Sony Computer Entertainment, Inc.

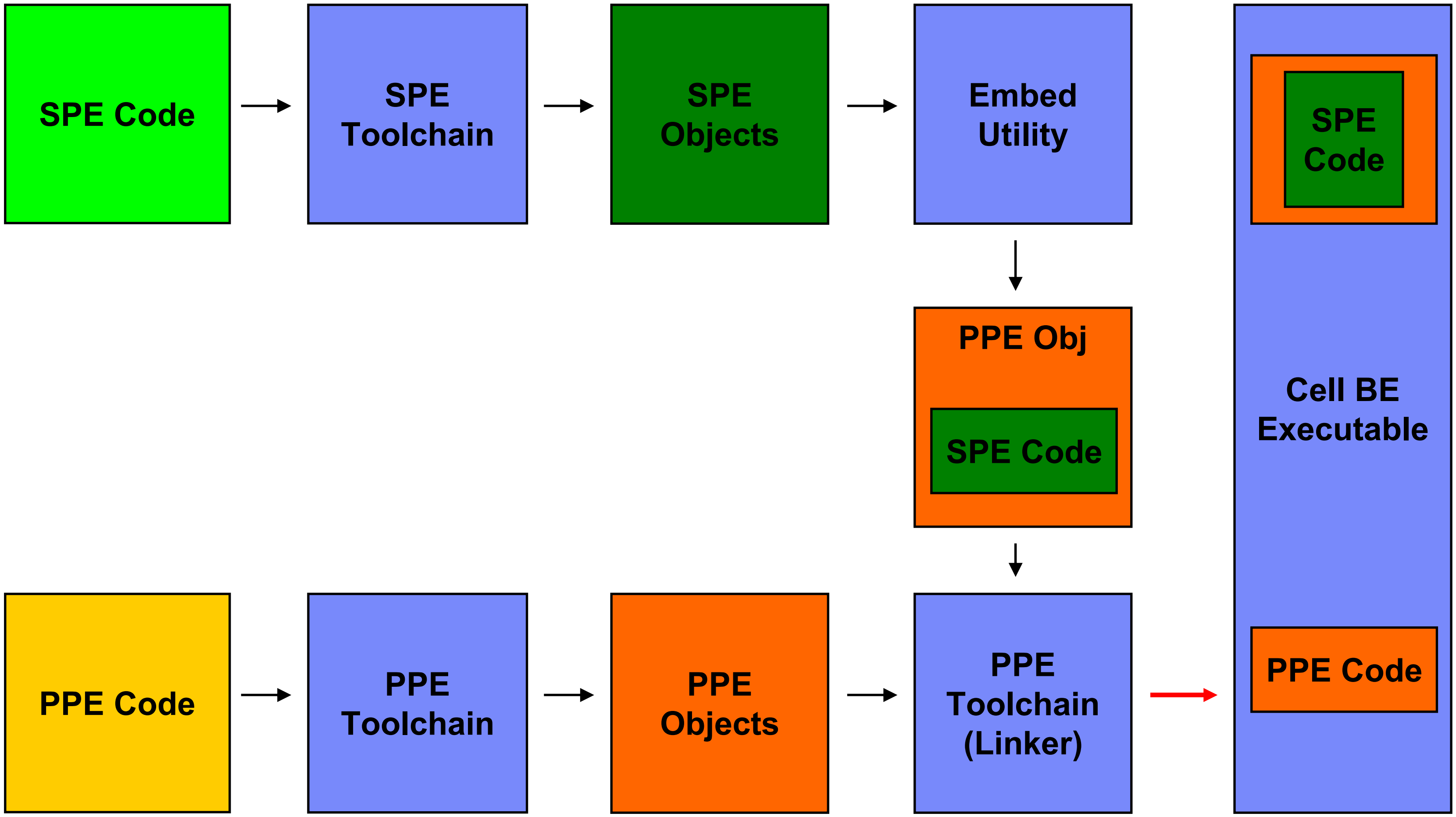
# How to build, compile and execute the “Hello World!” program

- **Pre-requisites**
  - Toolchain
  - Compiler
- **Build Process**
- **Source Code**
  - Makefiles
  - Source PPE
  - Source SPE
- **Simulator**
  - Getting the binary into the simulator
  - Running the binary

# The build process

# Build Process



# SDK 3.0 Makefile

# Compiling within the SDK

- **Top of build environment is /opt/cell/sdk/**
- **Includes the build environment files**
  - README_build_env.txt
    - Provides details on the build environment features, including files, structure and variables.
  - make.footer
    - Specifies all of the build rules needed to properly build CBEA binaries
    - Must be included in all SDK Makefiles (referenced relatively if $CELL_TOP is not defined)
    - Includes make.header
  - make.header
    - Specifies definitions needed to process the Makefiles
    - Includes make.env
  - make.env
    - Specifies the default compilers and tools to be used by make
- **make.footer and make.header should not be modified**

# Common Makefile variables

- **DIRS**
  - list of subdirectories to build first
- **PROGRAM_ppu PROGRAMS_ppu**
  - 32-bit PPU program (or list of programs) to build.
- **PROGRAM_ppu64 PROGRAMS_ppu64**
  - 64-bit PPU program (or list of programs) to build.
- **PROGRAM_spu PROGRAMS_spu**
  - SPU program (or list of programs) to build.
  - If written as a standalone binary, can run without being embedded in a PPU program.
- **LIBRARY_embed LIBRARY_embed64**
  - Creates a linked library from an SPU program to be embedded into a 32-bit or 64-bit PPU program.
- **OBJS OBJS_<program>**
  - List of objects for the programs (or one specific program). By default, all objects in the current directory are linked into the binary.
- **IMPORTS IMPORTS_<program>**
  - List of libraries to link in the programs (or one specific program). Also used by the PPU programs to embed the SPU linked library.

# Directory Layout and Examples of Makefile

- **sample**
  - sample.h
  - Makefile

```
DIRS = spu ppu
include $(CELL_TOP)/buildutils/make.footer
```

- **sample/spu**
  - Makefile
  - sample_spu.c

```
PROGRAM_spu = sample_spu
LIBRARY_embed = lib_sample_spu.a
include $(CELL_TOP)/buildutils/make.footer
```

- **sample/ppu**
  - Makefile
  - sample.c

```
PROGRAM_ppu = sample
IMPORTS = ../spu/lib_sample_spu.a
include $(CELL_TOP)/buildutils/make.footer
```

# Building The Code

- **Environment setup**
  - Set the CELL_TOP environment variable so that the makefile system can be found:
    - `export CELL_TOP=/opt/cell/sdk/`
    - make.footer contains the build rules for the makefile system
  - Ensure compilers or cross-compilers are in the executable search path
- **Separate SPE code and PPE code into different directories**
  - Each set of code has it's own makefile and toolchain to use
  - Suggestion: create a subdirectory called 'spu' in the directory where the PPU code is found
- **Makefile template for PPE code:**

```
DIRS = spu
PROGRAM_ppu = <PPU_executable_name>
IMPORTS = <spu_executable-embed.a> -lspe2
include $(CELL_TOP)/buildutils/make.footer
```

- **Makefile template for SPE code:**

```
PROGRAM_spu = <SPU_executable_name>
LIBRARY_embed = <spu_executable-embed.a>
include $(CELL_TOP)/buildutils/make.footer
```

# The “Hello World!” program

# Four Different Versions of “Hello World!”

- **PPE only**
- **SPE only**
- **Synergistic PPE and SPE: synchronous**
  - One SPE is used.
  - Main thread blocks and waits for the SPE code to run to completion
- **Synergistic PPE and SPE: asynchronous**
  - Eight SPEs are used
  - Main thread uses pthreads to get concurrent/asynchronous execution

# “Hello World!” – PPE Only

- **PPU program**
  - just like any “Hello World!” program one would write

```
#include <stdio.h>

int main(void)
{
        printf("Hello world!\n");
        return 0;
}
```

- **Makefile**
  - make.footer included to set up compiler and compiler flags
  - PROGRAM_ppu tells make to use PPC cross-compiler

**PROGRAM_ppu tells make to use PPC compiler**

```
PROGRAM_ppu = hello_ppu
include $(CELL_TOP)/buildutils/make.footer
```

# "Hello World!" – SPE Only

- **SPU Program**

```
#include <stdio.h>

int main()
{
        printf("Hello world!\n");
        return 0;
}
```

- **SPU Makefile**

**PROGRAM_spu tells make to use SPE compiler**

```
PROGRAM_spu   := hello_spu
include $(CELL_TOP)/buildutils/make.footer
```

# Synergistic PPE and SPE (SPE Embedded)

- Applications use software constructs called SPE contexts to manage and control SPEs.
- Linux schedules SPE contexts from all running applications onto the physical SPE resources in the system for execution according to the scheduling priorities and policies associated with the runable SPE contexts.
- libspe provides the means for communication and data transfer between PPE threads and SPEs.

# How does a PPE program start an SPE thread?

- 4 basic steps must be done by the PPE program
  - Create an SPE context.
  - Load an SPE executable object into the SPE context local store.
  - Run the SPE context. This transfers control to the operating system, which requests the actual scheduling of the context onto a physical SPE in the system.
  - Destroy the SPE context.

# SPE context creation

- **spe_context_create** - Create and initialize a new SPE context data structure.

```
#include <libspe2.h>

spe_context_ptr_t spe_context_create(unsigned int flags,
spe_gang_context_ptr_t gang)
```

  - *`flags`* - A bit-wise OR of modifiers that are applied when the SPE context is created.
  - *`gang`* - Associate the new SPE context with this gang context. If NULL is specified, the new SPE context is not associated with any gang.
  - On success, a pointer to the newly created SPE context is returned.

# spe_program_load

- **spe_program_load** - Load an SPE main program.

```
#include <libspe2.h>

int spe_program_load (spe_context_ptr_t spe,
spe_program_handle_t *program)
```

  - *spe* - A valid pointer to the SPE context for which an SPE program should be loaded.
  - *program* - A valid address of a mapped SPE program.

# spe_context_run

- **spe_context_run** - Request execution of an SPE context.

  ```
  #include &lt;libspe2.h&gt;
  ```

  ```
  int spe_context_run(spe_context_ptr_t spe, unsigned int *entry,
  unsigned int runflags, void *argp, void *envp, spe_stop_info_t
  *stopinfo)
  ```

  - *spe* - A pointer to the SPE context that should be run.
  - *entry* - Input: The entry point, that is, the initial value of the SPU instruction pointer, at which the SPE program should start executing. If the value of entry is SPE_DEFAULT_ENTRY, the entry point for the SPU main program is obtained from the loaded SPE image. This is usually the local store address of the initialization function crt0.
  - *runflags* - A bit mask that can be used to request certain specific behavior for the execution of the SPE context. 0 indicates default behavior.
  - *argp* - An (optional) pointer to application specific data, and is passed as the second parameter to the SPE program,
  - *envp* - An (optional) pointer to environment specific data, and is passed as the third parameter to the SPE program,
  - `stopinfo` An (optional) pointer to a structure of type spe_stop_info_t

# spe_context_destroy

- **spe_context_destroy** - Destroy the specified SPE context.

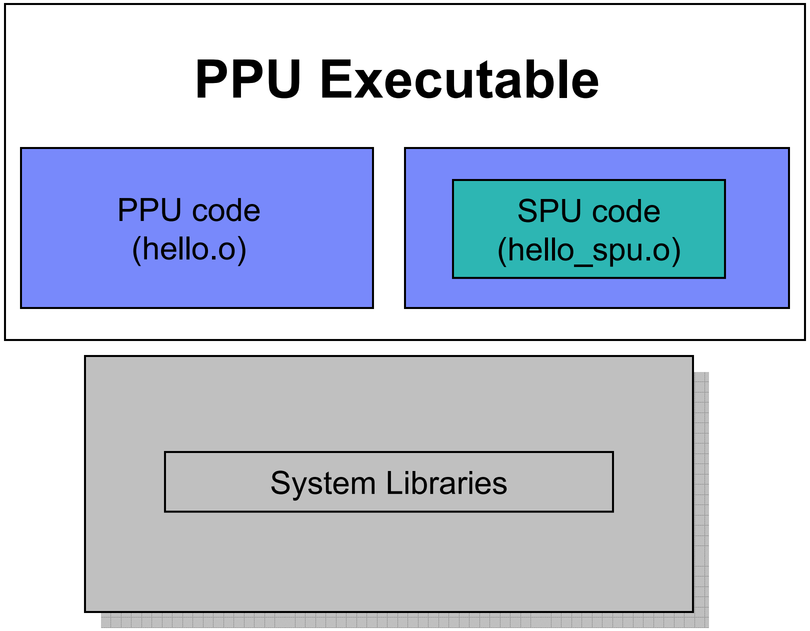
```
#include <libspe2.h>
int spe_context_destroy (spe_context_ptr_t spe)
```

  - *spe* - Specifies the SPE context to be destroyed
  - On success, **0** (zero) is returned, else -1 is returned

# “Hello World!” – PPE and SPE Combined Structure

- **SPU code**
  - Compiled with SPU specific toolchain
  - Object is repackaged as PPC ELF object
  - From this point forward normal PPU tools are used.
- **PPU code**
  - Compiled with normal PPU toolchain
- **Objects are linked to form a combined executable.**
- **At runtime, kernel extensions and SDK libraries are used to move the SPU code to an SPU and start the SPU thread.**



# "Hello World!" – Synergistic PPE and SPE (SPE Embedded)

- **SPU program**
  - Same as for SPE only
- **SPU Makefile**

```
PROGRAM_spu    := hello_spu
LIBRARY_embed   := hello_spu.a
include $(CELL_TOP)/buildutils/make.footer
```

# "Hello World!" – PPU program

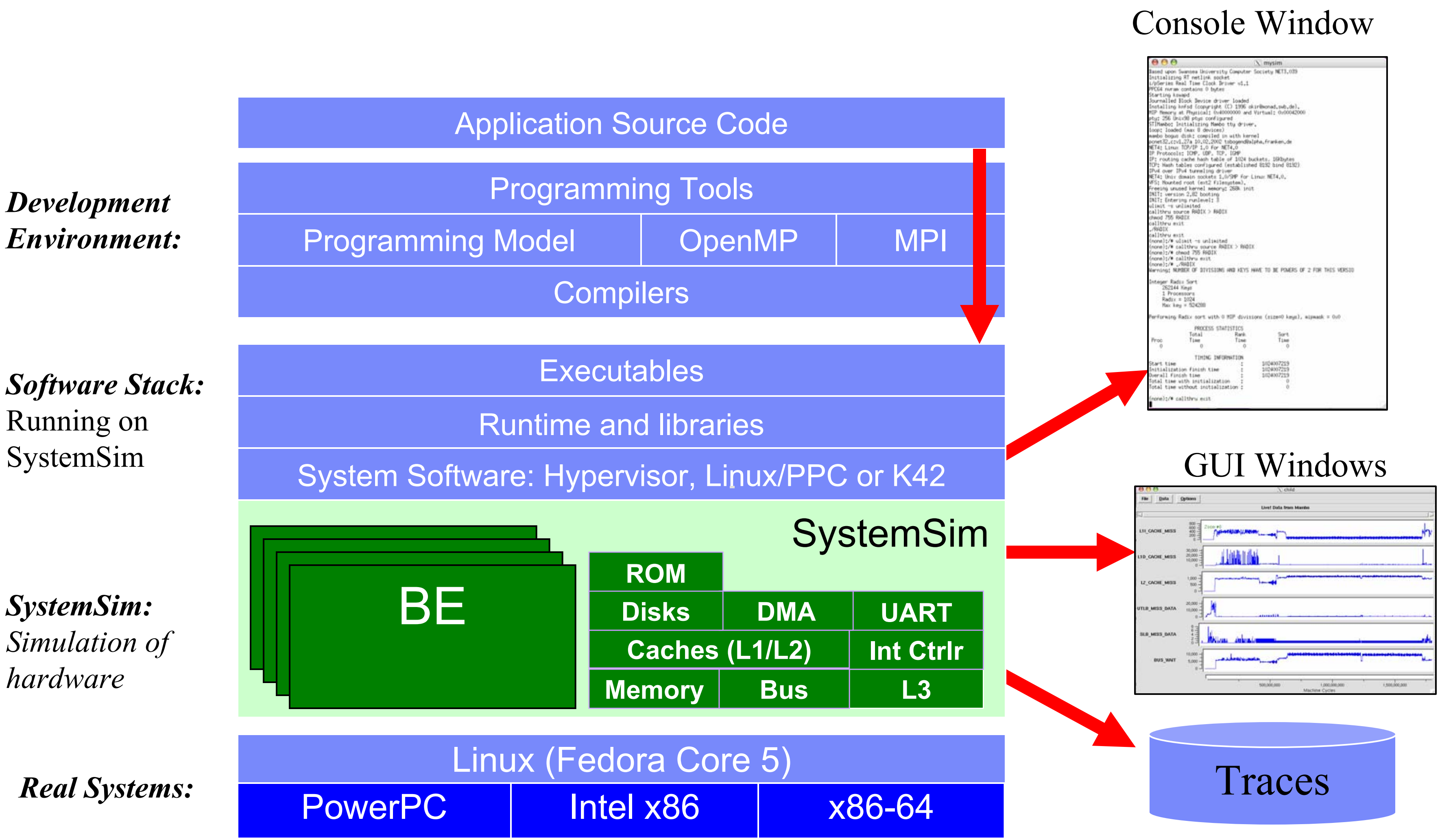
```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <libspe2.h>

extern spe_program_handle_t hello_spu;

int main(void)
{
    spe_context_ptr_t speid;
    unsigned int flags = 0;
    unsigned int entry = SPE_DEFAULT_ENTRY;
    void * argp = NULL;
    void * envp = NULL;
    spe_stop_info_t stop_info;
    int rc;

// Create an SPE context
    speid = spe_context_create(flags, NULL);
    if (speid == NULL) {
    perror("spe_context_create");
    return -2;
    }

// Load an SPE executable object into the
   SPE context local store
    if (spe_program_load(speid, &hello_spu))
   {
    perror("spe_program_load");
    return -3;
    }

// Run the SPE context
    rc = spe_context_run(speid, &entry, 0,
   argp, envp, &stop_info);
    if (rc < 0)
    perror("spe_context_run");

// Destroy the SPE context
    spe_context_destroy(speid);
    return 0;
}
```

**PPU Makefile**

```
DIRS = spu
PROGRAM_ppu = hello_be1
IMPORTS = spu/hello_spu.a -lspe2 -lpthread
include $(CELL_TOP)/buildutils/make.footer
```

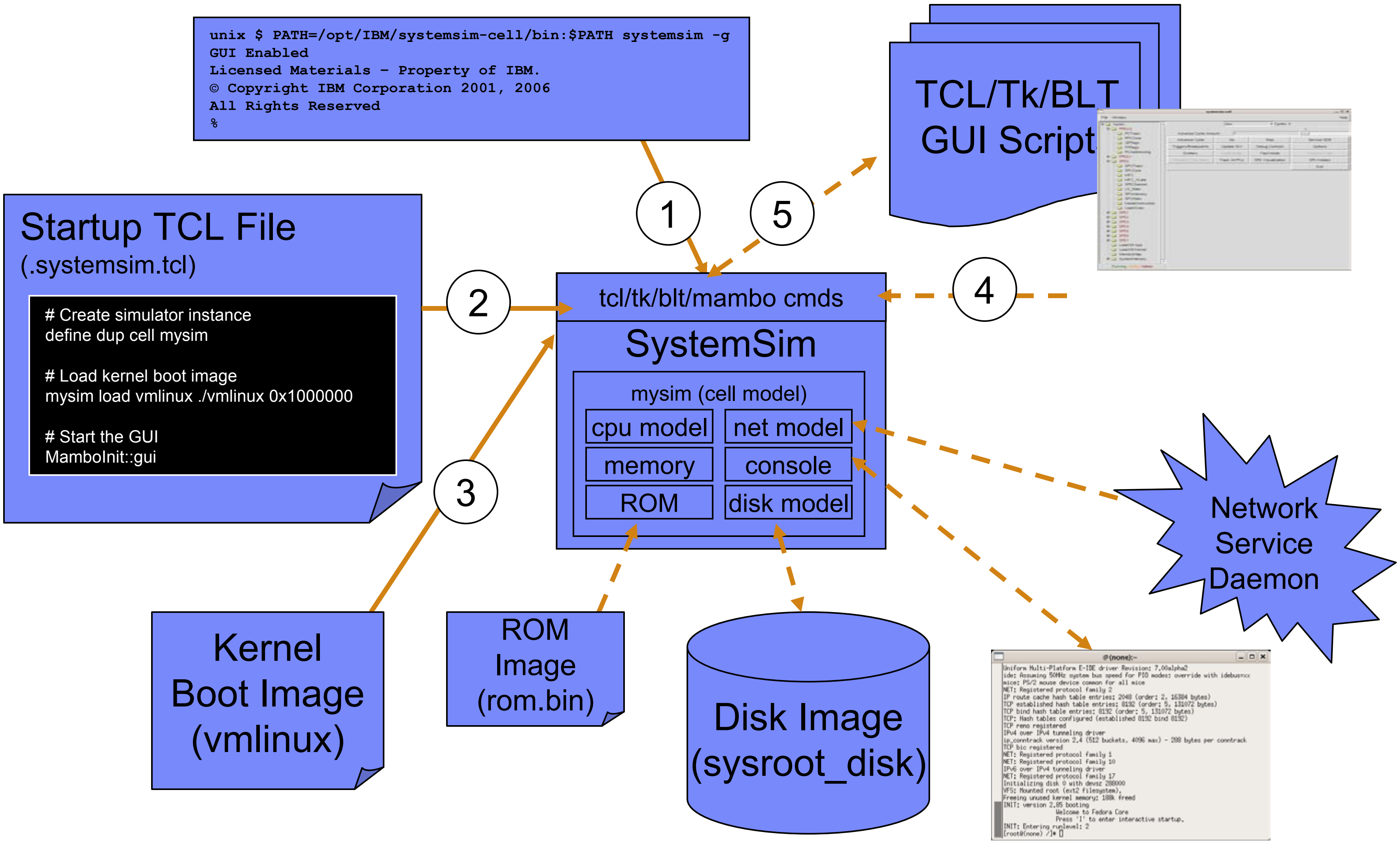
# The IBM Full System Simulator – An Overview

# Simulator Overview



# SystemSim Runtime Environment



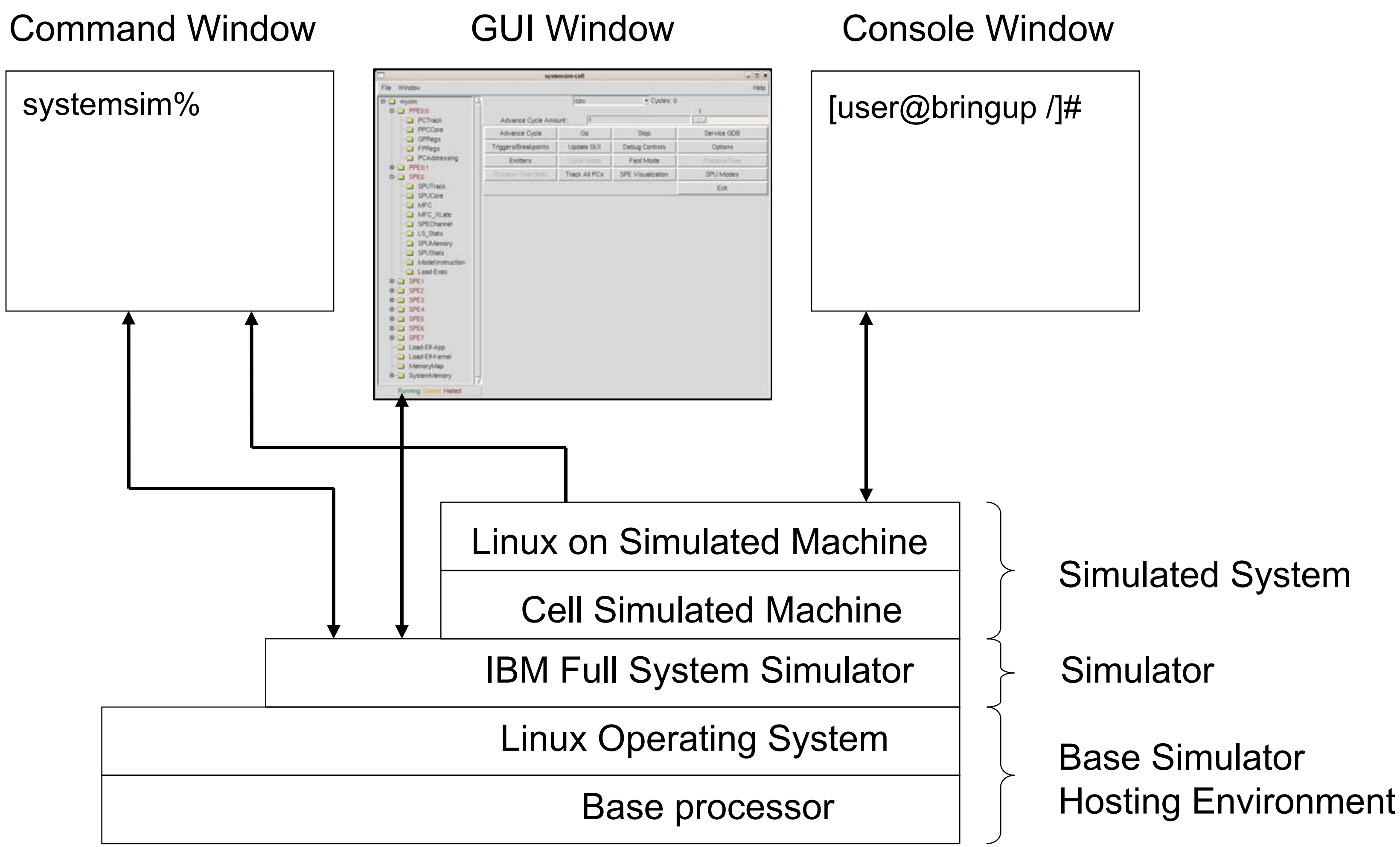
# SystemSim User Interface

- **Graphical interface**
  - Provides a visual display of the state of the simulated system, including the PPE and the eight (or 16) SPEs
  - Includes dialogs to view the contents of the registers, memory, and channels, and other architectural structures
  - Based on Tcl/Tk
  - Layered on top of the command line interface
- **Command line**
  - Uses Tcl (Tool Control Language) as the base command interpreter
  - All the standard Tcl commands are available
  - SystemSim commands to configure and create simulated machines
  - Commands (e.g. mysim) to control a specific simulated machine

# Operating-System Modes

- **Linux Mode**
  - Simulator boots a full Linux operating system on the simulated system
  - Applications are launched from the Linux console window and run
  - The simulated operating system handles all the system calls
- **Standalone Mode**
  - The application is loaded directly into the simulated machine without an operating system
  - The simulator traps all system calls made by the application and performs these functions in place of the operating system
  - Some restrictions apply, such as
    - The application must be statically linked with any libraries it needs
    - No virtual memory support is provided
    - Only a subset of system calls are supported

# Simulator Structure and Windows



# Interacting with the Simulator

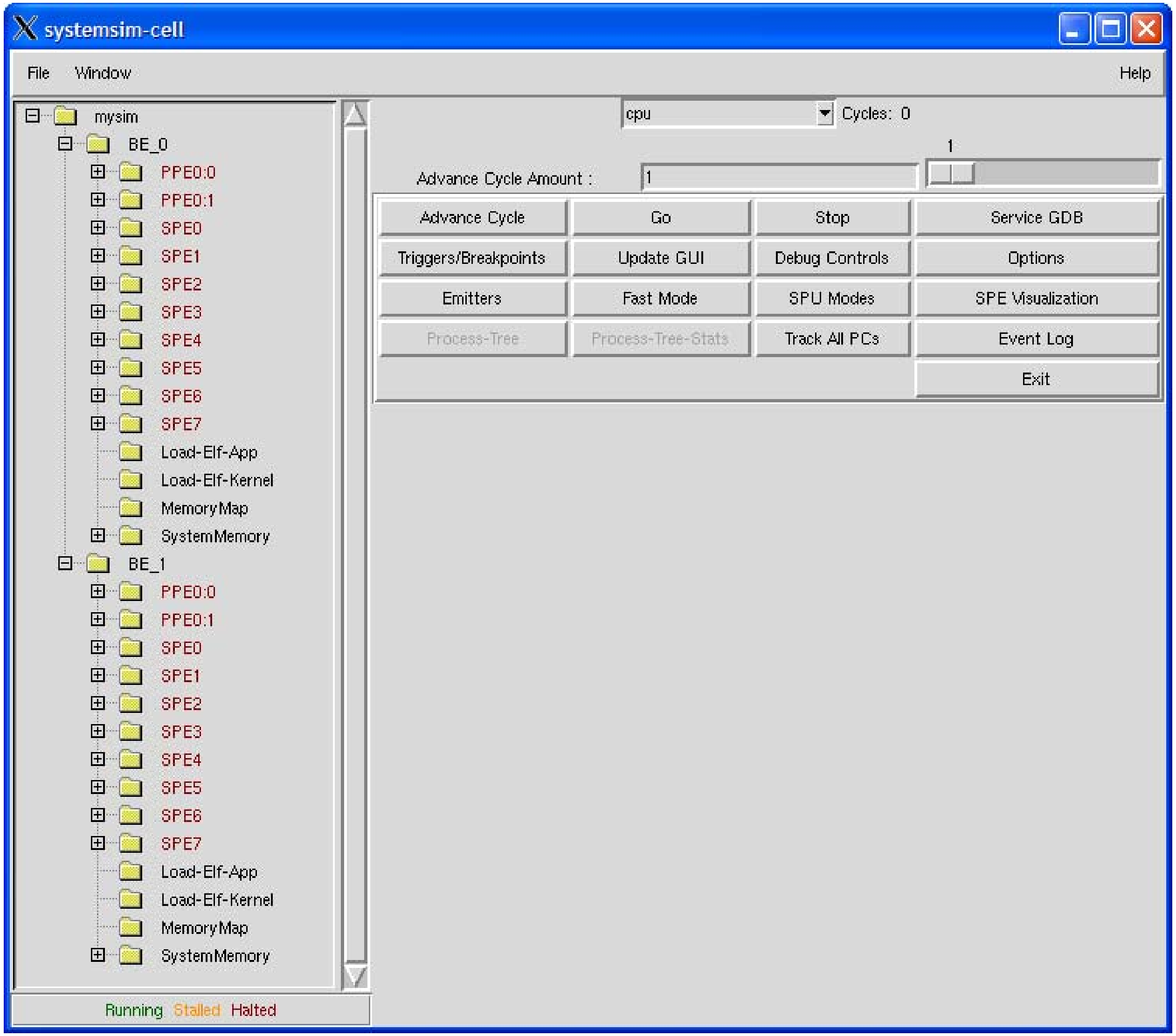
- **Issuing commands to the simulator**
  - in the simulator command window, or using the equivalent actions in the graphical user interface (GUI).
  - To control the simulator itself, configuring it to do such tasks as collect and display performance statistics on particular SPEs, or set breakpoints in code.
- **Issuing commands to the simulated system**
  - in the console window which is a Linux shell of the simulated Linux operating system.
  - The simulated system is the Linux environment on top of the simulated cell, where you run and debug programs.

# Starting the Simulator in GUI Interface

- **The simulator is invoked with the systemsim command "systemsim –g"**
  - Note: add /opt/ibm/systemsim-cell/bin to your path
- **Specify the initial run script using –f if configuration is needed**
  - file should be in the current directory or path qualified
  - This configures the simulated machine and prepares it for execution
  - The default is .systemsim.tcl
  - Samples are provided in the simulator run directory
    - Linux mode:
      - /opt/ibm/systemsim-cell/run/cell/linux/systemsim.tcl
- **Other systemsim options**
  - -n : do not open a console window
  - -q : suppress periodic run statistics messages
  - -g : enable the graphical interface
- **Starting the simulator in GUI mode with two Cell BE (SMP configuration)**
  - systemsim –g –f config_smp.tcl
- **Another way to start the simulator**
  - # cd /opt/ibm/systemsim-cell/run/cell/linux
  - #../run_gui

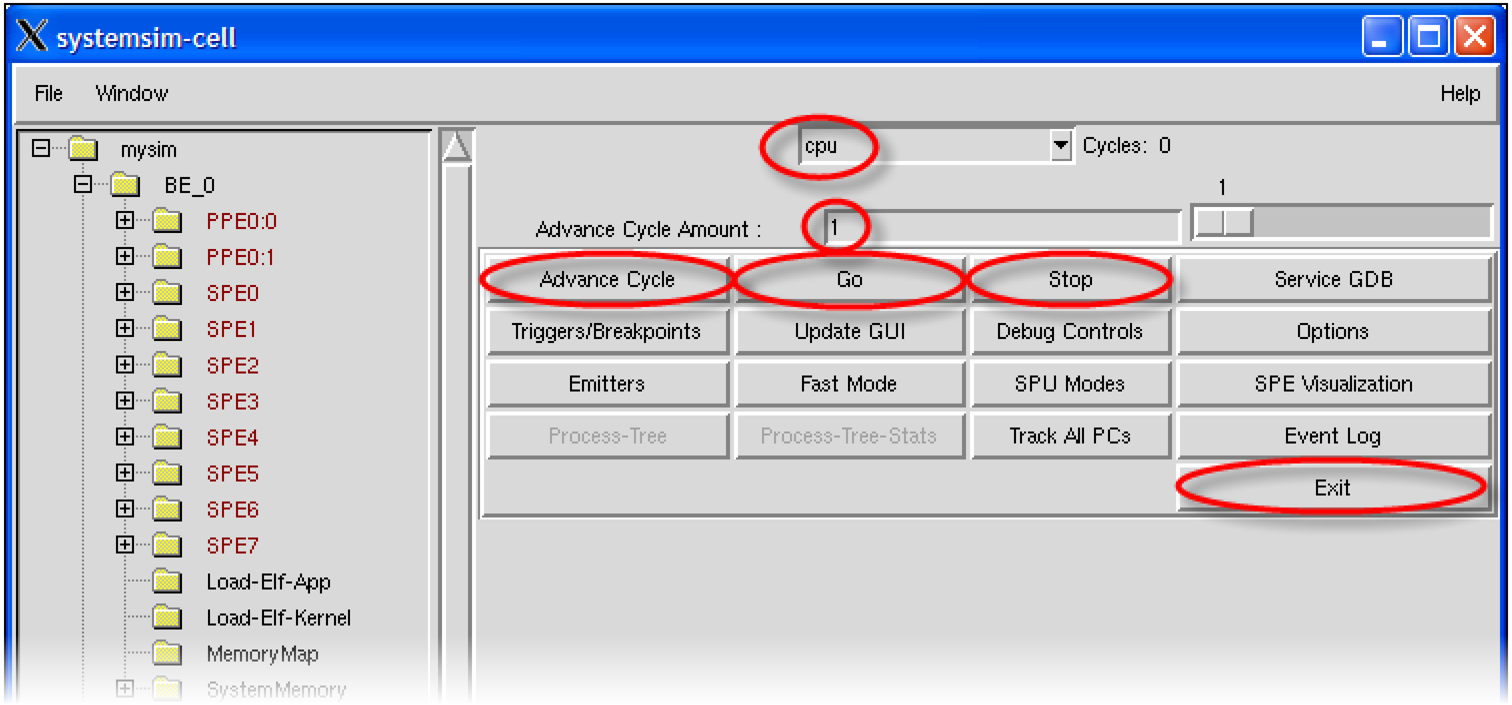
 

# SystemSim Cell GUI main panel



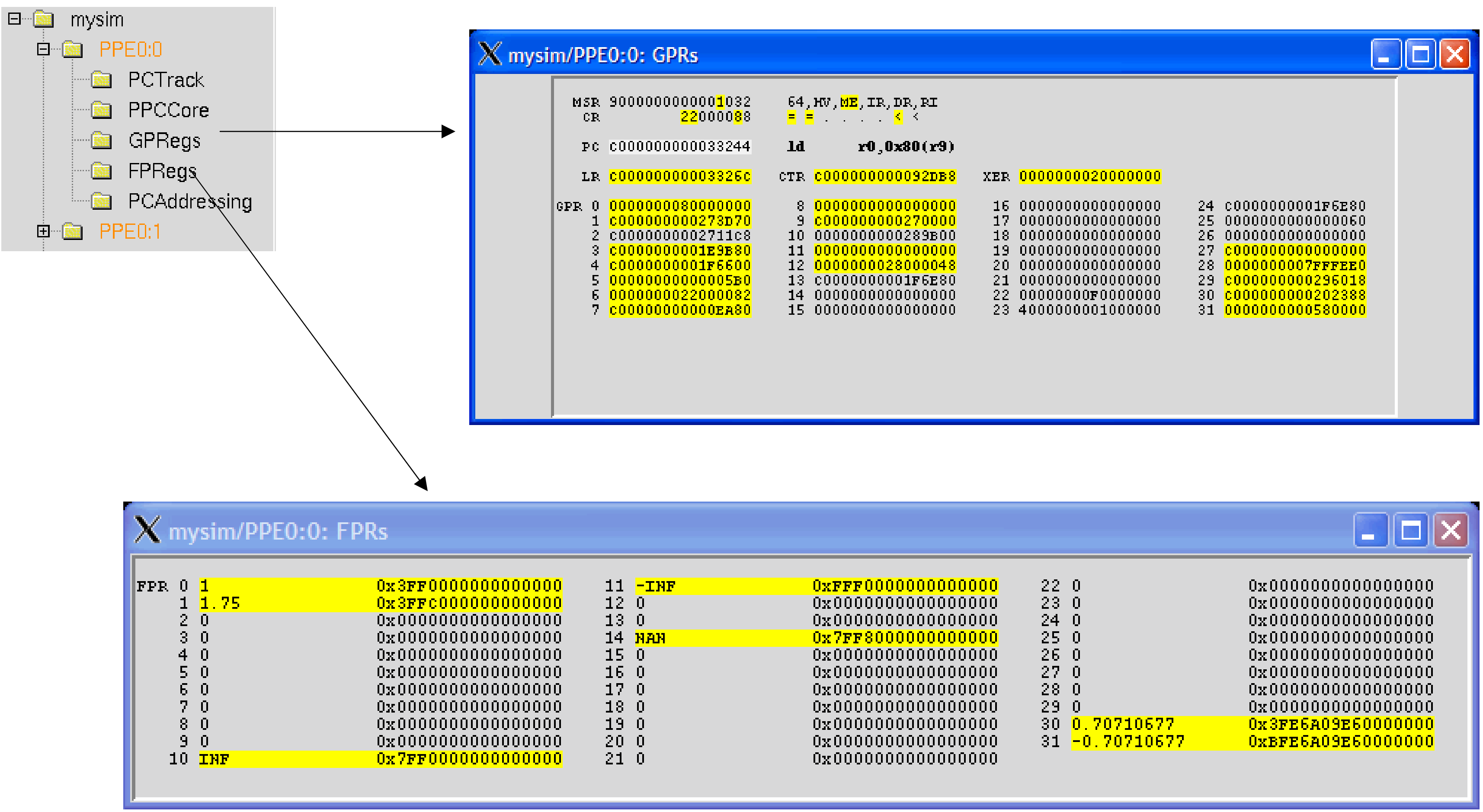

 

# Basic Simulator operations



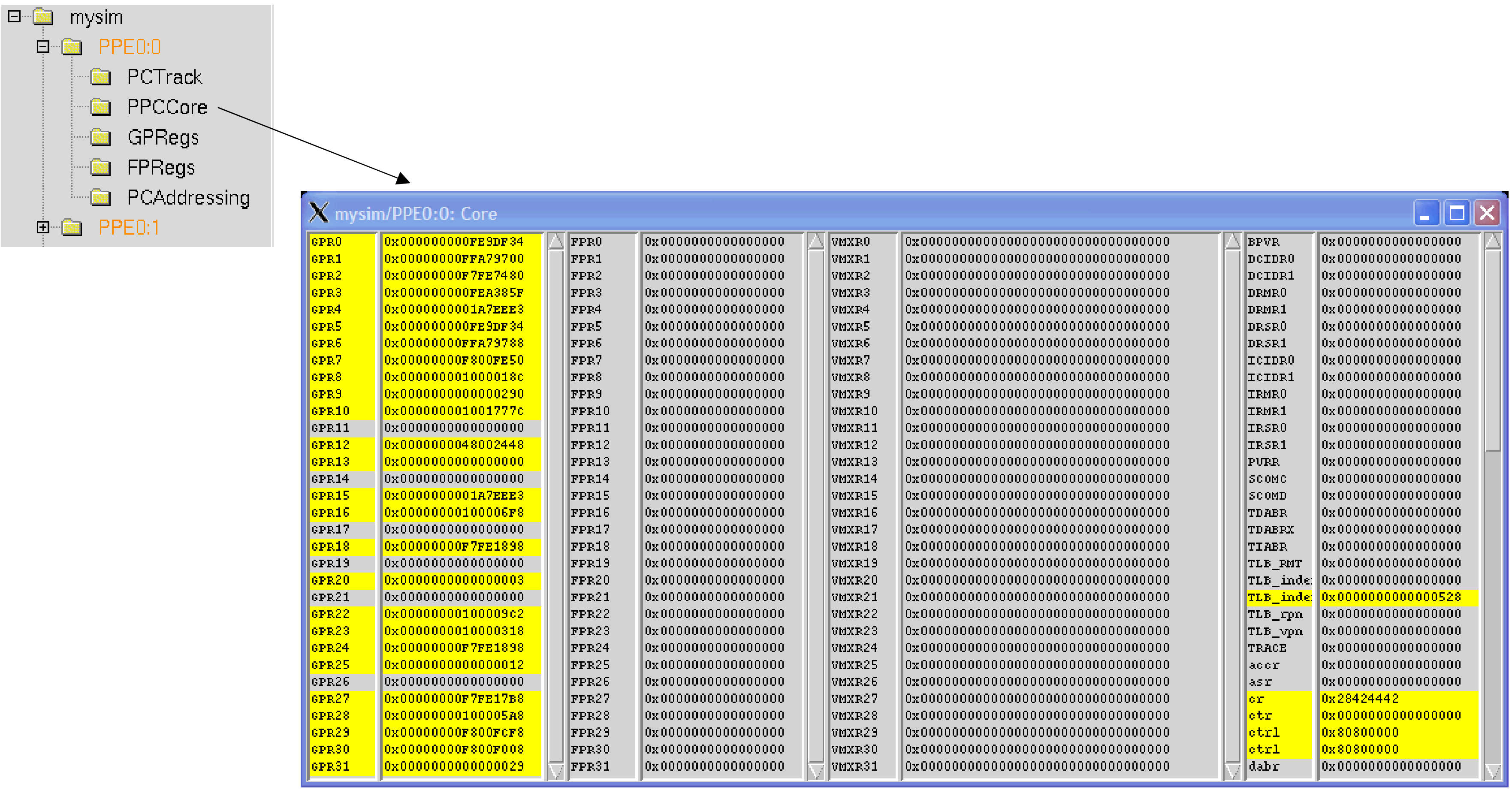

 

# The PPE


mysim
PPE0:0
PCTrack
PPCCore
GPRegs
FPRegs
PCAddressing
PPE0:1
mysim/PPE0:0: GPRs
MSR 9000000000001032 64,HV,ME,IR,DR,RI
CR 22000088 = = . . . . < <
PC C000000000033244 ld r0,0x80(r9)
LR C00000000003326C CTR C0000000000092DB8 XER 0000000020000000
GPR 0 0000000080000000 8 0000000000000000 16 0000000000000000 24 C0000000001F6E80
1 C000000000273D70 9 C000000000270000 17 0000000000000000 25 0000000000000060
2 C0000000002711C8 10 0000000000289B00 18 0000000000000000 26 0000000000000000
3 C0000000001E9B80 11 0000000000000000 19 0000000000000000 27 C000000000000000
4 C0000000001F6600 12 0000000028000048 20 0000000000000000 28 000000007FFFEE0
5 00000000000005B0 13 C0000000001F6E80 21 0000000000000000 29 C000000000296018
6 0000000022000082 14 0000000000000000 22 00000000F0000000 30 C000000000202388
7 C00000000000EA80 15 0000000000000000 23 4000000001000000 31 0000000000580000
mysim/PPE0:0: FPRs
FPR 0 1 0x3FF0000000000000 11 -INF 0xFFF0000000000000 22 0 0x0000000000000000
1 1.75 0x3FFC000000000000 12 0 0x0000000000000000 23 0 0x0000000000000000
2 0 0x0000000000000000 13 0 0x0000000000000000 24 0 0x0000000000000000
3 0 0x0000000000000000 14 NAN 0x7FF8000000000000 25 0 0x0000000000000000
4 0 0x0000000000000000 15 0 0x0000000000000000 26 0 0x0000000000000000
5 0 0x0000000000000000 16 0 0x0000000000000000 27 0 0x0000000000000000
6 0 0x0000000000000000 17 0 0x0000000000000000 28 0 0x0000000000000000
7 0 0x0000000000000000 18 0 0x0000000000000000 29 0 0x0000000000000000
8 0 0x0000000000000000 19 0 0x0000000000000000 30 0.70710677 0x3FE6A09E60000000
9 0 0x0000000000000000 20 0 0x0000000000000000 31 -0.70710677 0xBFE6A09E60000000
10 INF 0x7FF0000000000000 21 0 0x0000000000000000

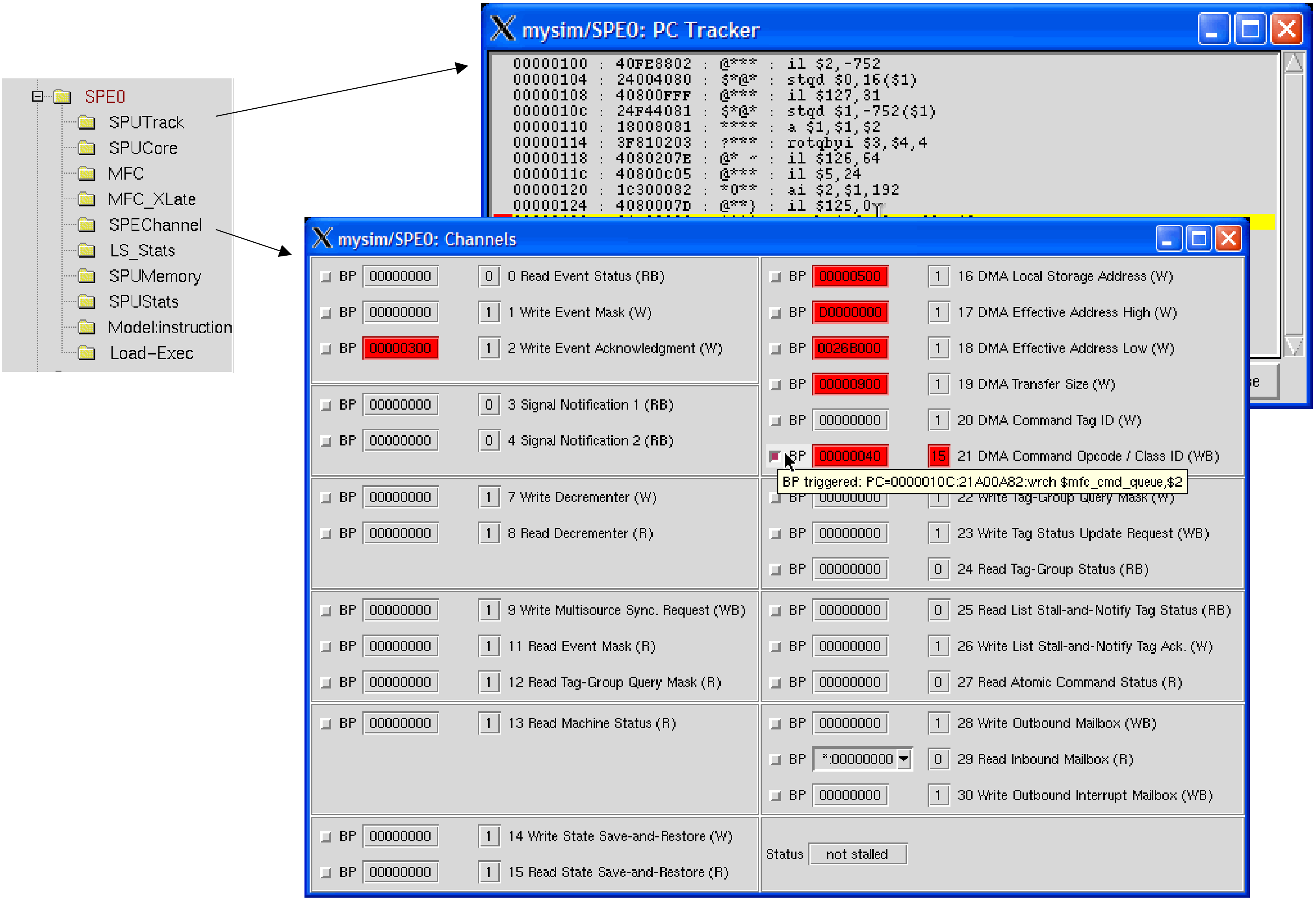
 

# The PPE


mysim
PPE0:0
PCTrack
PPCCore
GPRegs
FPRegs
PCAddressing
PPE0:1
mysim/PPE0:0: Core
GPR0 0x000000000FE9DF34 FPR0 0x0000000000000000 VMXR0 0x00000000000000000000000000000000 BPVR 0x0000000000000000
GPR1 0x00000000FFA79700 FPR1 0x0000000000000000 VMXR1 0x00000000000000000000000000000000 DCIDR0 0x0000000000000000
GPR2 0x00000000F7FE7480 FPR2 0x0000000000000000 VMXR2 0x00000000000000000000000000000000 DCIDR1 0x0000000000000000
GPR3 0x000000000FEA385F FPR3 0x0000000000000000 VMXR3 0x00000000000000000000000000000000 DRMR0 0x0000000000000000
GPR4 0x000000001A7EEE3 FPR4 0x0000000000000000 VMXR4 0x00000000000000000000000000000000 DRMR1 0x0000000000000000
GPR5 0x000000000FE9DF34 FPR5 0x0000000000000000 VMXR5 0x00000000000000000000000000000000 DRSR0 0x0000000000000000
GPR6 0x00000000FFA79788 FPR6 0x0000000000000000 VMXR6 0x00000000000000000000000000000000 DRSR1 0x0000000000000000
GPR7 0x00000000F800FE50 FPR7 0x0000000000000000 VMXR7 0x00000000000000000000000000000000 ICIDR0 0x0000000000000000
GPR8 0x000000001000018c FPR8 0x0000000000000000 VMXR8 0x00000000000000000000000000000000 ICIDR1 0x0000000000000000
GPR9 0x0000000000000290 FPR9 0x0000000000000000 VMXR9 0x00000000000000000000000000000000 IRMR0 0x0000000000000000
GPR10 0x000000001001777c FPR10 0x0000000000000000 VMXR10 0x00000000000000000000000000000000 IRMR1 0x0000000000000000
GPR11 0x0000000000000000 FPR11 0x0000000000000000 VMXR11 0x00000000000000000000000000000000 IRSR0 0x0000000000000000
GPR12 0x0000000048002448 FPR12 0x0000000000000000 VMXR12 0x00000000000000000000000000000000 IRSR1 0x0000000000000000
GPR13 0x0000000000000000 FPR13 0x0000000000000000 VMXR13 0x00000000000000000000000000000000 PURR 0x0000000000000000
GPR14 0x0000000000000000 FPR14 0x0000000000000000 VMXR14 0x00000000000000000000000000000000 SCOMC 0x0000000000000000
GPR15 0x000000001A7EEE3 FPR15 0x0000000000000000 VMXR15 0x00000000000000000000000000000000 SCOMD 0x0000000000000000
GPR16 0x00000000100006F8 FPR16 0x0000000000000000 VMXR16 0x00000000000000000000000000000000 TDABR 0x0000000000000000
GPR17 0x0000000000000000 FPR17 0x0000000000000000 VMXR17 0x00000000000000000000000000000000 TDABRX 0x0000000000000000
GPR18 0x00000000F7FE1898 FPR18 0x0000000000000000 VMXR18 0x00000000000000000000000000000000 TIABR 0x0000000000000000
GPR19 0x0000000000000000 FPR19 0x0000000000000000 VMXR19 0x00000000000000000000000000000000 TLB_RMT 0x0000000000000000
GPR20 0x0000000000000003 FPR20 0x0000000000000000 VMXR20 0x00000000000000000000000000000000 TLB_inde 0x0000000000000000
GPR21 0x0000000000000000 FPR21 0x0000000000000000 VMXR21 0x00000000000000000000000000000000 TLB_inde 0x0000000000000528
GPR22 0x00000000100009c2 FPR22 0x0000000000000000 VMXR22 0x00000000000000000000000000000000 TLB_rpn 0x0000000000000000
GPR23 0x0000000010000318 FPR23 0x0000000000000000 VMXR23 0x00000000000000000000000000000000 TLB_vpn 0x0000000000000000
GPR24 0x00000000F7FE1898 FPR24 0x0000000000000000 VMXR24 0x00000000000000000000000000000000 TRACE 0x0000000000000000
GPR25 0x0000000000000012 FPR25 0x0000000000000000 VMXR25 0x00000000000000000000000000000000 accr 0x0000000000000000
GPR26 0x0000000000000000 FPR26 0x0000000000000000 VMXR26 0x00000000000000000000000000000000 asr 0x0000000000000000
GPR27 0x00000000F7FE17B8 FPR27 0x0000000000000000 VMXR27 0x00000000000000000000000000000000 cr 0x28424442
GPR28 0x00000000100005A8 FPR28 0x0000000000000000 VMXR28 0x00000000000000000000000000000000 ctr 0x0000000000000000
GPR29 0x00000000F800FCF8 FPR29 0x0000000000000000 VMXR29 0x00000000000000000000000000000000 ctrl 0x80800000
GPR30 0x00000000F800F008 FPR30 0x0000000000000000 VMXR30 0x00000000000000000000000000000000 ctrl 0x80800000
GPR31 0x0000000000000029 FPR31 0x0000000000000000 VMXR31 0x00000000000000000000000000000000 dabr 0x0000000000000000

# The SPU


SPE0
SPUTrack
SPUCore
MFC
MFC_XLate
SPEChannel
LS_Stats
SPUMemory
SPUStats
Model:instruction
Load–Exec
mysim/SPE0: PC Tracker
00000100 : 40FE8802 : @*** : il $2,-752
00000104 : 24004080 : $*@* : stqd $0,16($1)
00000108 : 40800FFF : @*** : il $127,31
0000010c : 24F44081 : $*@* : stqd $1,-752($1)
00000110 : 18008081 : **** : a $1,$1,$2
00000114 : 3F810203 : ?*** : rotqbyi $3,$4,4
00000118 : 4080207E : @* ~ : il $126,64
0000011c : 40800c05 : @*** : il $5,24
00000120 : 1c300082 : *0** : ai $2,$1,192
00000124 : 4080007D : @**) : il $125,0
mysim/SPE0: Channels
BP 00000000 0 0 Read Event Status (RB)
BP 00000000 1 1 Write Event Mask (W)
BP 00000300 1 2 Write Event Acknowledgment (W)
BP 00000000 0 3 Signal Notification 1 (RB)
BP 00000000 0 4 Signal Notification 2 (RB)
BP 00000000 1 7 Write Decrementer (W)
BP 00000000 1 8 Read Decrementer (R)
BP 00000000 1 9 Write Multisource Sync. Request (WB)
BP 00000000 1 11 Read Event Mask (R)
BP 00000000 1 12 Read Tag-Group Query Mask (R)
BP 00000000 1 13 Read Machine Status (R)
BP 00000000 1 14 Write State Save-and-Restore (W)
BP 00000000 1 15 Read State Save-and-Restore (R)
BP 00000500 1 16 DMA Local Storage Address (W)
BP D0000000 1 17 DMA Effective Address High (W)
BP 0026B000 1 18 DMA Effective Address Low (W)
BP 00000900 1 19 DMA Transfer Size (W)
BP 00000000 1 20 DMA Command Tag ID (W)
BP 00000040 15 21 DMA Command Opcode / Class ID (WB)
BP triggered: PC=0000010C:21A00A82:wrch $mfc_cmd_queue,$2
BP 00000000 1 23 Write Tag Status Update Request (WB)
BP 00000000 0 24 Read Tag-Group Status (RB)
BP 00000000 0 25 Read List Stall-and-Notify Tag Status (RB)
BP 00000000 1 26 Write List Stall-and-Notify Tag Ack. (W)
BP 00000000 0 27 Read Atomic Command Status (R)
BP 00000000 1 28 Write Outbound Mailbox (WB)
BP *:00000000 0 29 Read Inbound Mailbox (R)
BP 00000000 1 30 Write Outbound Interrupt Mailbox (WB)
Status not stalled

# Simulator Modes – fast, simple, and cycle

- **The default simulation mode when the simulator starts is “simple”, or functional-only, simulation**
  - In this mode, the time / cycles to execute an application is NOT a meaningful indicator of execution time on real hardware
- **To get meaningful performance results:**
  - Select “Cycle” mode on the GUI
  - Enter “mysim mode cycle” in the command window
- **This will make the simulator run slower**
  - Depending on the workload, simulation time could increase by 10x to 100x
  - But you can switch between modes as needed, so you can limit this overhead to just the relevant portions of the simulation

# How to Exchange Files between Host and Simulator

- **callthru**
  - A command issued from a simulated windows (from the simulator)
  - “backdoor” communication mechanism for the simulated environment to communicate with the host environment
  - Useful for bringing in files to the simulated environment without shutting down and restarting the simulator
  - Example: (binary host → simulator)
    - callthru source /opt/cell_class/Hands-on-30/hello/hello_ppu/hello_ppu > hello_ppu
    - chmod 755 hello_ppu
    - ./hello_ppu
  - Example (result file simulator → host)
    - callthru sink /home/systemsim-cell/results/result_file < cat result_file
    - exporting result files out of the simulated environment for later inspection

# Execute Binary

- From the simulated windows, bring executable into the simulator by using the callthru utility, e.g.,
  - callthru source /opt/cell_class/Hands-on-30/hello/hello_ppu/hello_ppu > hello_ppu
- Execute binary
  - chmod 755 hello_ppu
  - ./hello_ppu

**Tip!**
**Copy binary to /tmp/´<exe> on host to shorten the filename**

# Building three types of the hello world! program

# Directory Structure

/opt/cell_class/Hands-on-30/hello

- hello_ppu
- hello_spu
- hello_be1 synchronous spu thread (hello_be1-sync)
  - spu
- hello_be1 asynchronous spu thread (hello_be1-async)
  - spu

# Hands-on Exercise

1. Create a directory hello_ppu, write a hello world ppu program and create a Makefile, then compile and execute it as a standalone ppu program
2. Create a directory hello_spu, write a hello world spu program and create a Makefile, then compile and execute it as a standalone spu program
3. Create a directory hello_be1, and write a ppu program that calls an spu program to write hello world in a synchronous manner. Create all ppu and spu makefiles. Compile and execute those programs to demonstrate the basic structure of a simple PPE-SPE software synergy model (PPE-single SPE model)
4. Same as in 3. but with asynchronous thread
5. Producing a simple multi-threaded hello world program
   - See instructions in the next page

**Need to compile (use make) and run the executables on the simulator**

# Hands-on – multi-threaded hello world

To produce a simple program for the CBE, you should follow the steps listed below (this example is included in the SDK in /opt/cell/sdk/src/tutorial/simple).

The project is called simple. For this example, the PPE code will be built in the project directory, instead of a ppu sub-directory.

This program creates SPE threads that output "Hello Cell (#)\n" to the systemsim output window, where # is the spe_id of the SPE thread that issued the print.

**1. Create a directory named simple**.

**2. In directory simple, create a file named Makefile using the following code**:

```
################################################################################
# Subdirectories
################################################################################
DIRS := spu
################################################################################
# Target
################################################################################
PROGRAM_ppu := simple
################################################################################
# Local Defines
################################################################################
```

# Hands-on – multi-threaded hello world (cont'd)

```
IMPORTS := spu/lib_simple_spu.a -lspe2 -lpthread
# imports the embedded simple_spu library
# allows consolidation of spu program into ppe binary
################################################################################
# make.footer
################################################################################
# make.footer is in the top of the SDK
ifdef CELL_TOP
include $(CELL_TOP)/buildutils/make.footer
else
include ../../../../buildutils/make.footer
Endif
```

**3. In directory simple, create a file simple.c using the following code:**

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <libspe2.h>
#include <pthread.h>
```

# Hands-on – multi-threaded hello world (cont’d)

```
extern spe_program_handle_t simple_spu;
#define MAX_SPU_THREADS 16
void *ppu_pthread_function(void *arg) {
spe_context_ptr_t ctx;
unsigned int entry = SPE_DEFAULT_ENTRY;
ctx = *((spe_context_ptr_t *)arg);
if (spe_context_run(ctx,&entry, 0, NULL, NULL, NULL) < 0) {
perror ("Failed running context");
exit (1);
}
pthread_exit(NULL);
}
int main()
{
int i,spu_threads;
spe_context_ptr_t ctxs[MAX_SPU_THREADS];
pthread_t threads[MAX_SPU_THREADS];
```

# Hands-on – multi-threaded hello world (cont'd)

```
/* Determine the number of SPE threads to create */
spu_threads = spe_cpu_info_get(SPE_COUNT_USABLE_SPES, -1);
if (spu_threads > MAX_SPU_THREADS) spu_threads = MAX_SPU_THREADS;
/* Create several SPE-threads to execute 'simple_spu' */
for(i=0; i<spu_threads; i++) {
/* Create context */
if ((ctxs[i] = spe_context_create (0, NULL)) == NULL) {
perror ("Failed creating context");
exit (1);
}
/* Load program into context */
if (spe_program_load (ctxs[i],&simple_spu)) {
perror ("Failed loading program");
exit (1);
}
/* Create thread for each SPE context */
if (pthread_create (&threads[i], NULL,&ppu_pthread_function,&ctxs[i])) {
perror ("Failed creating thread");
exit (1);
}
```

# Hands-on – multi-threaded hello world (cont'd)

```
/* Wait for SPU-thread to complete execution. */
for (i=0; i<spu_threads; i++) {
if (pthread_join (threads[i], NULL)) {
perror("Failed pthread_join");
exit (1);
}
}
printf("\nThe program has successfully executed.\n");
return (0);
}
```

**4. Create a directory named spu.**

# Hands-on – multi-threaded hello world (cont’d)

**5. In the directory spu, create a file named Makefile using the following code:**

```
#####################################################################################
# Target
######################################################################################
PROGRAMS_spu := simple_spu
# created embedded library
LIBRARY_embed := lib_simple_spu.a
######################################################################################
# make.footer
######################################################################################
# make.footer is in the top of the SDK
ifdef CELL_TOP
include $(CELL_TOP)/buildutils/make.footer
else
include ../../../../../buildutils/make.footer
endif
```

# Hands-on – multi-threaded hello world (cont'd)

**6. In the same directory, create a file simple_spu.c using the following code:**

```
#include <stdio.h>
int main(unsigned long long id)
{
/* The first parameter of an spu program will always be the spe_id of the spe
* thread that issued it.
*/
printf("Hello Cell (0x%llx)\n", id);
return 0;
}
```

**7. Compile the program by entering the following command at the command line while in the simple directory:**

```
make
```

# Summary

- **Compile and execute different types of cell programs on the simulator**
  - Understand the basic differences between a ppu, spu, and BE program
  - Understand the embedded concept of a cellBE program
  - Understand the build process
  - Understand the contents of different Makefile
  - Understand the basic operations of the simulator

# Special Notices -- Trademarks

This document was developed for IBM offerings in the United States as of the date of publication. IBM may not make these offerings available in other countries, and the information is subject to change without notice. Consult your local IBM business contact for information on the IBM offerings available in your area. In no event will IBM be liable for damages arising directly or indirectly from any use of the information contained in this document.

Information in this document concerning non-IBM products was obtained from the suppliers of these products or other public sources. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. Send license inquires, in writing, to IBM Director of Licensing, IBM Corporation, New Castle Drive, Armonk, NY 10504-1785 USA.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

The information contained in this document has not been submitted to any formal IBM test and is provided "AS IS" with no warranties or guarantees either expressed or implied.

All examples cited or described in this document are presented as illustrations of the manner in which some IBM products can be used and the results that may be achieved. Actual environmental costs and performance characteristics will vary depending on individual client configurations and conditions.

IBM Global Financing offerings are provided through IBM Credit Corporation in the United States and other IBM subsidiaries and divisions worldwide to qualified commercial and government clients. Rates are based on a client's credit rating, financing terms, offering type, equipment type and options, and may vary by country. Other restrictions may apply. Rates and offerings are subject to change, extension or withdrawal without notice.

IBM is not responsible for printing errors in this document that result in pricing or information inaccuracies.

All prices shown are IBM's United States suggested list prices and are subject to change without notice; reseller prices may vary.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

Many of the features described in this document are operating system dependent and may not be available on Linux. For more information, please check: http://www.ibm.com/systems/p/software/whitepapers/linux_overview.html

Any performance data contained in this document was determined in a controlled environment. Actual results may vary significantly and are dependent on many factors including system hardware configuration and software design and configuration. Some measurements quoted in this document may have been made on development-level systems. There is no guarantee these measurements will be the same on generally-available systems. Some measurements quoted in this document may have been estimated through extrapolation. Users of this document should verify the applicable data for their specific environment.

Revised January 19, 2006

# Special Notices (Cont.) -- Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States and/or other countries: alphaWorks, BladeCenter, Blue Gene, ClusterProven, developerWorks, e business(logo), e(logo)business, e(logo)server, IBM, IBM(logo), ibm.com, IBM Business Partner (logo), IntelliStation, MediaStreamer, Micro Channel, NUMA-Q, PartnerWorld, PowerPC, PowerPC(logo), pSeries, TotalStorage, xSeries; Advanced Micro-Partitioning, eServer, Micro-Partitioning, NUMACenter, On Demand Business logo, OpenPower, POWER, Power Architecture, Power Everywhere, Power Family, Power PC, PowerPC Architecture, POWER5, POWER5+, POWER6, POWER6+, Redbooks, System p, System p5, System Storage, VideoCharger, Virtualization Engine.

A full list of U.S. trademarks owned by IBM may be found at: http://www.ibm.com/legal/copytrade.shtml.

Cell Broadband Engine and Cell Broadband Engine Architecture are trademarks of Sony Computer Entertainment, Inc. in the United States, other countries, or both.
Rambus is a registered trademark of Rambus, Inc.
XDR and FlexIO are trademarks of Rambus, Inc.
UNIX is a registered trademark in the United States, other countries or both.
Linux is a trademark of Linus Torvalds in the United States, other countries or both.
Fedora is a trademark of Redhat, Inc.
Microsoft, Windows, Windows NT and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries or both.
Intel, Intel Xeon, Itanium and Pentium are trademarks or registered trademarks of Intel Corporation in the United States and/or other countries.
AMD Opteron is a trademark of Advanced Micro Devices, Inc.
Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States and/or other countries.
TPC-C and TPC-H are trademarks of the Transaction Performance Processing Council (TPPC).
SPECint, SPECfp, SPECjbb, SPECweb, SPECjAppServer, SPEC OMP, SPECviewperf, SPECapc, SPEChpc, SPECjvm, SPECmail, SPECimap and SPECsfs are trademarks of the Standard Performance Evaluation Corp (SPEC).
AltiVec is a trademark of Freescale Semiconductor, Inc.
PCI-X and PCI Express are registered trademarks of PCI SIG.
InfiniBand™ is a trademark the InfiniBand® Trade Association
Other company, product and service names may be trademarks or service marks of others.

Revised July 23, 2006

# Special Notices - Copyrights



IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY 12533-6351

The IBM home page is http://www.ibm.com
The IBM Microelectronics Division home page is
http://www.chips.ibm.com